Performance and energy optimization of concurrent pipelined applications

Anne Benoit, Paul Renaud-Goud, Yves Robert
LIP, Ecole Normale Supérieure de Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07, France
Université de Lyon - UMR 5668 - CNRS - ENSL - UCBL - INRIA
{Anne.Benoit|Paul.Renaud-Goud|Yves.Robert}@ens-lyon.fr

Abstract*—In this paper, we study the problem of finding optimal mappings for several independent but concurrent workflow applications, in order to optimize performance-related criteria together with energy consumption. Each application consists in a linear chain graph with several stages, and processes successive data sets in pipeline mode, from the first to the last stage. We study the problem complexity on different target execution platforms, ranking from fully homogeneous platforms to fully heterogeneous ones. The goal is to select an execution speed for each processor, and then to assign stages to processors, with the aim of optimizing several concurrent optimization criteria. There is a clear trade-off to reach, since running faster and/or more processors leads to better performance, but the energy consumption is then very high. Energy savings can be achieved at the price of a lower performance, by reducing processor speeds or enrolling fewer resources. We consider two mapping strategies: in one-to-one mappings, a processor is assigned a single stage, while in interval mappings, a processor may process an interval of consecutive stages of the same application. For both mapping strategies and all platform types, we establish the complexity of several multi-criteria optimization problems, whose objective functions combine period, latency and energy criteria. In particular, we exhibit cases where the problem is NP-hard with concurrent applications, while it can be solved in polynomial time for a single application. Also, we demonstrate the difficulty of performance/energy trade-offs by proving that the tri-criteria problem is NP-hard, even with a single application on a fully homogeneous platform.

***Index Terms*—workflow; pipeline; complexity results; period; latency; energy.**

I. Introduction

In this paper, we aim at optimizing the execution of several independent pipelined applications that execute concurrently on a given platform. Indeed, pipelined applications are becoming increasingly prevalent, see for instance [1]–[3]. Mapping such applications onto parallel platforms is a challenging problem, that becomes even more difficult when platforms are heterogeneous (nowadays a standard assumption). Another level of difficulty is added when considering several independent applications which are executed concurrently on the platform and that compete for available resources.

We focus in this work on pipelined applications with the regular structure of a linear chain. Such applications are ubiquitous in streaming environments, as for instance video and audio encoding and decoding, DSP applications, image processing, and so on [1]–[5]. Furthermore, the regularity of these applications render them amenable to a high-level parallel programming approach based on algorithmic skeletons [6], [7]. Skeletons ease the task of the application developer and make it easy to tailor his/her specific problem to a target platform. In linear pipelined applications, a series of data sets enter the input stage and progress from stage to stage until the final result is computed. Each stage has its own communication and computation requirements: it reads an input from the previous stage, processes the data and outputs a result to the next stage. Each data set is first input to the first stage, and final results are output from the last stage. The pipeline operates in synchronous mode: after a transient behavior due to the initialization delay, a new data set is completed every period.

Typical performance-related objectives for such pipelined operations are the *period* (which is defined as the inverse of the throughput) or the *latency* (also called response time) [5], [8]–[13]. Formally, the period of a mapping is defined as the time interval required between the beginning of the execution of two consecutive data sets. The period is dictated by the critical resource: it is equal to the longest cycle time of a processor. For instance under a strict one-port communication model with no overlap of communications and computations, it is the sum of the time to perform all incoming communications, the time to perform all outgoing communications, and the total computation time. As for the latency, it is defined as the time elapsed between the beginning and the end of the execution of a given data set, hence it measures the response time of the system to process the data set entirely. Period and latency already are conflicting objectives when executing a single application. When several applications run concurrently, the scheduler must decide which resources to select and assign to each application, so that all users receive a fair share of the platform.

In the recent years, another critical problem arose, namely the *energy consumption* of computational platforms. As an example, the Earth Simulator requires about 12 MW (Mega Watts) of peak power, and PetaFlop systems may require 100 MW of power, nearly the output of a small power plant (300 MW). At \$100 per MW.Hour, peak operation of a PetaFlop machine may thus cost \$10,000 per hour [14]. Current estimates state that cooling costs \$1 to \$3 per watt of heat dissipated [15]. This is just one of the many economical reasons why energy-aware scheduling has proved to be an important issue in the past decade, even without considering battery-powered systems such as laptops and embedded systems.

The emphasis of this paper is on a multi-criteria approach, where efficient trade-offs must be found between performance-related objectives that are typical of pipelined applications, namely period and latency minimization, and the total energy consumed by enrolled resources. For this purpose, we consider multi-modal processors: each processor has a discrete number of speeds (or modes) of computation, which can be obtained by changing the processor frequency: the faster the speed, the less efficient energetically-speaking [16]. At the beginning of execution, we must decide at which speed each computer will operate, and this speed is then fixed for the whole execution. The energy-oriented objective is to minimize the total energy consumption of the platform; it is computed as the sum of the energy consumed by each processor, which is a function of its speed (dynamic energy) and of a fixed overhead (static energy), similarly to the model in [17].

Our global aim is to execute all applications efficiently while minimizing the energy consumed. Unfortunately, the goals of low power consumption and efficient scheduling are contradictory. Indeed, period and/or latency can be minimized by using more energy to speed up processors, while energy can be minimized by reducing processor speeds, hence performance-related objectives. How to deal with these contradictory objective functions? In traditional approaches, one would form a linear combination of the different objectives and treat the result as the new objective to be optimized. But is it natural for the user to maximize the quantity 0.7P + 0.3E, where P is the period and E the energy? Since criteria are very different in nature, it does not make much sense for a user to make a linear combination of them. Thus we advocate the use of multi-criteria mappings with thresholds. Now, each criteria combination can be handled in a natural and meaningful way: one single criterion is optimized, under the condition that a threshold is enforced for all other criteria. This leads to two interesting questions. If we fix energy, we get the *laptop problem*, which asks "What is the best schedule achievable using a particular energy budget, before battery becomes critically low?" Fixing schedule quality gives the *server problem*, which asks "What is the least energy required to achieve a desired level of performance?"

The optimization problem can then be stated as follows: given a set of applications and a computational platform, which stage to assign to which processor? We consider two different mapping strategies: *one-to-one* mappings, for which each application stage is allocated to a distinct processor; and *interval* mappings, where each participating processor is assigned an interval of consecutive stages. These mapping strategies have been widely used in the literature when mapping one single application (see [8], [9], [12]), and we extend them naturally to the mapping of several concurrent applications without allowing any processor sharing. This assumption is quite realistic from the point of view of the platform manager whose goal may be to secure an efficient (albeit concurrent) execution for each application, and is further motivated from a theoretical point of view in Section III-C.

We target three different platform types: *fully homogeneous* platforms have identical processors and interconnection links; *communication homogeneous* platforms have identical links but different-speed processors, thus introducing a first degree of heterogeneity; and finally, *fully heterogeneous* platforms, with different-speed processors and different capacity links, constitute the most difficult problem instance.

Finally, we aim at optimizing several contradictory criteria, namely period, latency and energy, and we study all combination of these criteria. However, when taking energy into account, we always include the period in the combination, since the energy is an energy spent per time unit, which makes sense only in a pipelined execution of the application, while latency by its own takes only one single data set into account. Thus we consider two mono-criterion problems consisting in minimizing the period or the latency, and then two bi-criteria problems combining period/latency or period/energy, and finally the tri-criteria problem combining all three optimization criteria. Altogether, with two mapping strategies, three target platforms and five criteria combination, we have 30 problems to solve. A major contribution of this paper is to establish the complexity of all these problems in the context of multiple concurrent pipelined applications.

The problem of mapping a single linear chain application onto parallel platforms in order to minimize latency and/or period has already been widely studied, in particular on homogeneous platforms (see the pioneering papers [8] and [9]) and later for heterogeneous platforms (see [12], [13]). These results focus on the mapping of one single application, while we add the complexity of satisfying several users who each have different requirements for their applications. We were able to extend polynomial time algorithms to this multi-application setting, and to exhibit cases in which the problem becomes NP-hard because of this additional difficulty. Of course, problem instances which were already NP-hard with a single application remain difficult with several concurrent applications. Moreover, we consider a new and important objective function, namely energy minimization, and this is the first study (to the best of our knowledge) which combines performance-related objectives with energy in the context of pipelined applications. As expected, combining all three criteria (period, latency and energy) leads to even more difficult optimization problems: the problem is NP-hard even with a single application on a fully homogeneous platform.

The paper is organized as follows. We start by illustrating and motivating the problem with a simple example in Section II. Then we describe the framework in Section III. The next two sections constitute the heart of the paper: we assess the complexity of all problem instances. Results for period or latency minimization are reported in Section IV, while results for multi-criteria problems are presented in Section V. Finally we conclude in Section VI.

II. Motivating example

In this small example, we have two applications and three processors, as shown on Figure 1. We restrict to interval mappings, where a processor can be assigned only a set of consecutive stages of a single application. The first stage of App_1 receives a data of size 1, then computes 3 operations, and finally sends a data of size 3 to the second stage, and so on. If both stages are assigned to the same processor, there will be no communication cost to pay; otherwise this cost will depend on the communication volume (3 in this case) and on the link bandwidth between the corresponding processor pair. For the computational platform, each processor has two execution modes. For instance, P_1 can process 3 operations per time unit in its first mode, and 6 in its second one, against 6 or 8 for P_2, and 1 or 6 for P_3. The energy consumption of a processor is equal to the square of its speed, which is quite a realistic assumption (see Section III-E for more details on the model for energy consumption). Finally, all communication link bandwidths are set to 1.

We compute the global period as follows: $T = \max(T_1, T_2)$, where T_i is the period of the i^{th} application ($i = 1, 2$). The global latency is defined in a similar way, as the maximum of the latency achieved by all applications. Note that when the energy is not a criterion to minimize, all processors can run in their higher modes (as fast as possible), because this can only improve the performance-related criteria (period and latency). In this case, either a processor is used at its fastest speed, or it is turned off. In order to minimize the period without energy constraints, we map the whole first application onto processor P_3, the first half of the second application onto processor P_2, and the rest onto processor P_1. The period is then:

$$\max\left(\max\left(\frac{1}{1}, \frac{3+2+1}{6}, \frac{0}{1}\right), \right. \\ \left. \max\left(\max\left(\frac{0}{1}, \frac{2+6}{8}, \frac{1}{1}\right), \max\left(\frac{1}{1}, \frac{4+2}{6}, \frac{1}{1}\right)\right)\right) = 1 \quad (1)$$

Equation (1) reads as follows: we compute the cycle-time of each processor as the maximum time spent for incoming communications, computations, and outgoing communications, thus considering a model in which communications and computations are overlapped. We then take the maximum of these quantities to derive the period. Note that the cycle-time of each processor is exactly 1 and there is no idle time on computation, thus it is not possible to achieve a better period: this mapping is optimal for the period minimization problem.

The minimum latency is obtained by removing all communications and using the fastest processors. A mapping that returns the optimal latency (in the absence of other criteria) is for instance the one which maps the first application on P_1 and the second application on P_2, thus achieving a global latency of:

$$\max\left(\frac{1}{1} + \frac{3+2+1}{6} + \frac{0}{1}, \frac{0}{1} + \frac{2+6+4+2}{8} + \frac{1}{1}\right) = 2.75 \quad (2)$$

In Equation (2), we simply compute the longest execution path for each application. The bottleneck is the second application, and we cannot achieve a better latency since we pay no communication and use the fastest processor for this application. This latency is thus optimal.

The minimum energy is obtained when we use fewer processors, each running in slowest mode. Since we assume that a processor cannot be assigned stages of two different applications, two processors are required in the example. For instance, we can map the first application on P_1 running in its lowest mode and the second application on P_3 running in its lowest mode too, thus achieving an energy of $3^2 + 1^2 = 10$. This is the minimum energy consumption required to run both applications. We observe that the period is then:

$$\max\left(\max\left(\frac{1}{1}, \frac{3+2+1}{3}, \frac{0}{1}\right), \max\left(\frac{0}{1}, \frac{2+6+4+2}{1}, \frac{1}{1}\right)\right) = 14$$

As expected, running at a slower pace to save energy leads to poorer performances. Trade-offs must be found when considering several antagonistic optimization criteria.

For instance, if we try to minimize the energy consumption under the constraint that the period is not greater than 2, we can use the first mode of each processor. Then the first application is mapped onto P_1, the first three stages of the second application are mapped onto P_2 and its last stage is mapped onto P_3. The global period is 2, and the consumed energy is $3^2 + 6^2 + 1^2 = 46$. This may be quite a reasonable compromise between energy and period: indeed, with the mapping minimizing the period (period of 1), the energy consumption was $6^2 + 8^2 + 6^2 = 136$.

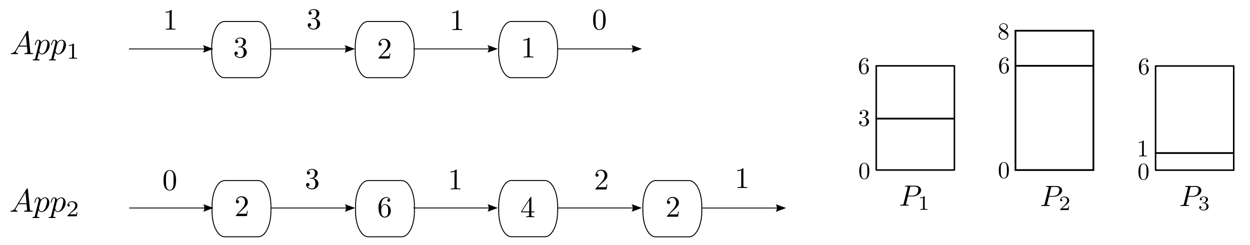

Figure 1. Example with two applications and three multi-modal processors.

III. Framework

We start with a formal description of the applicative framework (Section III-A) and the target execution platform (Section III-B). Next in Section III-C, we introduce and motivate the mapping strategies. We are then ready to formally describe the performance objective criteria (period and latency) in Section III-D, and then to finally discuss the energy model in Section III-E.

A. Applicative framework

We consider A independent application workflows ($A \geq 1$) to be executed concurrently; each application operates on a collection of data sets that are executed in a pipelined fashion. For $1 \leq a \leq A$, let n_a be the number of stages of application a, and $N = \sum_{a=1}^{A} n_a$ be the total number of stages. For $1 \leq k \leq n_a$, δ_a^k is the size of the output data of $\mathcal{S}_a^k$, the k^{th} stage of application a and w_a^k is its computation requirement. The first stage $\mathcal{S}_a^1$ of each application, $1 \leq a \leq A$, receives an input of size δ_a^0 from the outside world, while the last stage of each application $\mathcal{S}_a^{n_a}$ returns the result (of size $\delta_a^{n_a}$) to the outside world.

B. Target platform

The target platform is composed of p processors, which are fully interconnected; there is a bidirectional link $P_u \leftrightarrow P_v$ between any processor pair P_u and P_v, of bandwidth $b_{u,v}$. For simplification, we assume that $2A$ additional processors $P_{\mathsf{in}_1}, \dots, P_{\mathsf{in}_A}$ and $P_{\mathsf{out}_1}, \dots, P_{\mathsf{out}_A}$ are devoted to input/output operations of the applications (in fact these additional processors are virtual processes that may well be shared by the same physical resource). Initially, for each $a \in \{1, \dots, A\}$, the input data for each task of the application a resides on P_{in_a}, while all results must be returned to and stored on P_{out_a}. These special processors are all connected to the p processors of the target platform.

We use a linear cost model for communications; it takes $X/b_{u,v}$ time units to send (resp. receive) a message of size X to (resp. from) P_v. With the mapping rules that we enforce (see Section III-C below), it turns out that a processor never has to perform two concurrent ingoing nor outgoing communications: at any time-step, a processor is involved in at most one send, one computation and one receive. However, these three operations can either be parallel (as in the example of Section II) or serialized. With parallel operations, we have the *overlap* model that corresponds to multi-threaded communication libraries such as MPICH2 [18]. With sequential operations, we have the *no-overlap* model that is well-suited to single-threaded programs.

Processors are multi-modal: every processor P_u is associated with a set of speeds $\mathsf{S_u} = \{s_{u,1}, \dots, s_{u,m_u}\}$. During the mapping process, we need to choose one speed in $\mathsf{S_u}$ for each processor P_u that is enrolled, and this speed is fixed during the whole execution.

Then we classify particular cases which are important, both from a theoretical and practical perspective. *Fully homogeneous platforms* have identical processors (all processors have a common speed set: $\mathsf{S_u} = \mathsf{S}$) and homogeneous communication devices ($b_{u,v} = b$ for all link bandwidths). They represent typical parallel machines. *Communication homogeneous platforms* are still interconnected with homogeneous communication devices, but they may have processors with different speed sets ($\mathsf{S_u} \neq \mathsf{S_v}$). They correspond to networks of workstations with plain TCP/IP interconnects or other LANs. *Fully heterogeneous platforms* are the most general, fully heterogeneous architectures. Hierarchical platforms made up with several clusters interconnected by slower backbone links can be modeled this way.

C. Mapping strategies and scheduling

We consider two mapping strategies, *one-to-one* and by *interval*. *One-to-one* mappings obey the simplest rule: each application stage is allocated to a distinct processor. While easier to optimize and implement, this rule may be unduly restrictive, and is likely to pay high communication costs. Obviously, it also requires that $p \geq N$, thereby limiting its applicability to larger platforms (or fewer and smaller applications). A natural extension is to search for *interval* mappings, where each participating processor is assigned an interval of consecutive stages. Intuitively, assigning several consecutive stages to the same processors will increase their computational load, but may well dramatically decrease communication requirements. Interval mappings have been widely used in the literature, see [3], [5], [8], [9], [12] among others.

We point out that both one-to-one and interval mappings forbid any processor sharing, or re-use, across applications. We could introduce *general* mappings that would allow any processor to execute any number of stages, consecutive or not, taken from one or several applications. However, there are several reasons, both practical and theoretical, to restrict to interval mappings:

- On the practical side, we envision a computer center where applications, or jobs, cannot share resources because of security rules or of batch-assignment procedures. The goal of the platform manager is to secure an efficient (albeit concurrent) execution for each application (performance-related criteria) while minimizing the energy consumption of the whole platform.
- On the theoretical side, there are two problems with general mappings:
  1) they immediately lead to NP-hard optimization problems, even for the simplest mono-criterion problem: period minimization for a single application mapped onto homogeneous and uni-modal processors, paying no communication cost (straightforward reduction from 2-PARTITION);
  2) they lead to intricate scheduling problems for period/latency bi-criteria problems.

The latter problem is the most important, although we discovered it only quite recently [19]. Basically, even when the mapping is given, scheduling the execution is a problem of combinatorial nature. With general mappings, a processor typically has several incoming and/or outgoing communications, and it is difficult to orchestrate these operations so as to minimize conflicting objectives such as period and latency. This holds true both for the overlap and no-overlap models. On the contrary, with interval mappings, we have two key properties:

1) the execution graph is acyclic, meaning that data leaving one processor never returns to that processor;
2) each processor has at most one incoming and one outgoing communication.

Once the mapping has been determined, these two properties allow for a straightforward scheduling: each operation is executed as soon as possible.

D. Performance optimization criteria

We are now ready to formally define the *period* and the *latency* of one-to-one and interval mappings. Because there is no processor sharing, we can focus on a single application.

An interval mapping is a partition of the set of stages $\mathcal{S}^1$ to $\mathcal{S}^n$ into m intervals $I_j = [d_j, e_j]$ such that $d_j \leq e_j$ for $1 \leq j \leq m$, $d_1 = 1$, $d_{j+1} = e_j + 1$ for $1 \leq j \leq m-1$ and $e_m = n$. Then, the function $\mathsf{al} : [1,n] \mapsto [1,p]$ associates a processor number to each stage number. In a one-to-one mapping, this function is a one-to-one assignment. In an interval mapping, for $1 \leq j \leq m$, the whole interval I_j is mapped onto the same processor $P_{\mathsf{al}(d_j)}$, i.e., for $d_j \leq i \leq e_j$, $\mathsf{al}(i) = \mathsf{al}(d_j)$. Also, two intervals (from the same application or from two different applications) cannot be mapped onto the same processor, i.e., for $1 \leq j, j' \leq m$, $j \neq j'$, $\mathsf{al}(d_j) \neq \mathsf{al}(d_{j'})$.

The period of this single application is expressed in the overlap model as:

$$T^{(overlap)} = \max_{j \in \{1,\dots,m\}} \left(\max \left(\frac{\delta^{d_j - 1}}{b_{\mathsf{al}(d_j-1),\mathsf{al}(d_j)}}, \frac{\sum_{i=d_j}^{e_j} w^i}{s_{\mathsf{al}(d_j)}}, \frac{\delta^{e_j}}{b_{\mathsf{al}(d_j),\mathsf{al}(e_j+1)}} \right) \right) \quad (3)$$

The maximum in the previous expression is replaced by a sum when considering the no-overlap model, since all operations are serialized. The period is then:

$$T^{(no-overlap)} = \max_{j \in \{1,\dots,m\}} \left(\frac{\delta^{d_j - 1}}{b_{\mathsf{al}(d_j-1),\mathsf{al}(d_j)}} + \frac{\sum_{i=d_j}^{e_j} w^i}{s_{\mathsf{al}(d_j)}} + \frac{\delta^{e_j}}{b_{\mathsf{al}(d_j),\mathsf{al}(e_j+1)}} \right) \quad (4)$$

The latency is the time to process a single data entirely, so it is identical in both communication models:

$$L = \frac{\delta^0}{b_{\mathsf{al}(0),\mathsf{al}(1)}} + \sum_{j=1}^{m} \left(\sum_{i=d_j}^{e_j} \frac{w^i}{s_{\mathsf{al}(d_j)}} + \frac{\delta^{e_j}}{b_{\mathsf{al}(d_j),\mathsf{al}(e_j+1)}} \right) \quad (5)$$

Again, the simplicity of Equations (3), (4) and (5) is a very useful property of interval mappings, and greatly simplifies the solution of multi-criteria problems.

These are the period and latency of one single application, and we need to define a global period and latency function to be optimized. The simplest approach is to minimize $X = \max_{a \in \{1,\dots,A\}} (X_a)$, where X_a is the period or latency of application a, for $a \in \{1,\dots,A\}$. However, the concurrent applications can be of completely different nature and/or economic value, so that their periods or latencies are not always comparable. Therefore we aim at minimizing

$$X = \max_{a \in \{1,\dots,A\}} W_a \times X_a, \quad (6)$$

where $W_a > 0$ is a weight associated to each application and X_a is the period or latency of application a, for $a \in \{1,\dots,A\}$. W_a can be 1 (we retrieve a simple maximum) or a priority ratio (fixed by the platform manager and/or paid by the user). We can also let $W_a = 1/X_a^*$, where X_a^* is the objective function computed when the application is executed alone on the platform; in this case $W_a \times X_a$ represents the slowdown factor of application a, and X corresponds to the maximum stretch [20].

E. Energy model

The last criterion is the energy consumption of the platform, which is defined as the sum of the energy $E(u)$ consumed by each processor P_u enrolled in the mapping. We assume that $E(u)$ consists of a static part and of a dynamic part: $E(u) = E_{stat}(u) + E_{dyn}(s_u)$. The static part $E_{stat}(u)$ is the static cost for a processor to be in service, and does not depend on the speed s_u at which the processor is running. On the contrary, the dynamic part $E_{dyn}(s_u)$ is of the form $E_{dyn}(s) = s^\alpha$, where $\alpha > 1$ is an arbitrary rational number. It is sometimes assumed that $\alpha = 2$ [17], as we did in the example of Section II, but all our results hold for any value of α.

The energy $E(u)$ is an energy consumed per time unit, so it must be associated with a duration. However, the execution of a pipelined application with arbitrarily many consecutive data sets may last for an unbounded amount of time. Hence we always consider a combination of energy and period objective criteria, because the latency by its own takes only one single data set into account, and does not reflect a pipelined execution.

IV. Complexity of mono-criterion problems

Table I summarizes all mono-criterion complexity results. Each column corresponds to a platform type: **proc-hom** denotes identical speed processors while **proc-het** represents heterogeneous processors; **com-hom** means identical communication links, while they differ for **com-het**. We also report results for the case **special-app**, which corresponds to applications whose stages are all identical (all w_a^k are equal), and no communication cost is paid (all δ_a^k are equal to 0).

First, note that since we do not consider energy minimization issues in our mono-criterion optimization problems, we can systematically run processors at their highest speed, and thus use classical results established in a context with no energy. Most NP-completeness proofs come from the single

	proc-hom **com-hom**	**proc-het** **special-app**	**com-hom**	**com-het**
Period - *one-to-one*	polynomial (binary search)			NP-complete
Period - *interval*	polynomial (dyn. prog. + greedy)	NP-complete(*)	NP-complete	
Latency - *one-to-one*	polynomial	NP-complete(*)		NP-complete
Latency - *interval*	polynomial (binary search)			NP-complete

Table I
COMPLEXITY RESULTS FOR MONO-CRITERION OPTIMIZATION PROBLEMS.

application problem which already was NP-hard, see [12], [21] for the proofs. The two special entries denoted with (*) are problem instances which could be solved in polynomial time for a single application, but becomes NP-hard with several ones. Remaining entries correspond to polynomial algorithms that were already existing for a single application and that have been extended for several ones. Finally, all results apply to both the overlap and no-overlap models, and to all objective functions introduced in Section III-D: more precisely, polynomial problems remain polynomial for arbitrary weights W_a in Equation (6), while NP-complete problems are already difficult with $W_a = 1$. Due to space limitation, we report here only the proofs for the period minimization problems. The other proofs are provided in the companion research report [22].

A. One-to-one mappings

Theorem 1: On communication homogeneous platforms, a one-to-one mapping that minimizes the period can be determined in polynomial time.

Proof: The following proof is an adaptation of the algorithm described in [12], which finds the minimum period under the same hypothesis but for a single application. The main idea remains the same, since on communication homogeneous platforms the application that the stage belongs to does not matter for a one-to-one mapping.

The optimal period belongs to the set:

$$\mathcal{T} = \left\{ W_a \times \max\left(\frac{\delta_a^{k-1}}{b}, \frac{w_a^k}{s_u}, \frac{\delta_a^k}{b}\right), \right.$$
$$\left. 1 \le a \le A, 1 \le k \le n_a, 1 \le u \le p \right\}$$

because it is equal to the product of W_a by the cycle-time of some processor P_u, running in its fastest mode s_u, and executing one of the N stages, S_a^k. First we compute the set $\mathcal{T}$ and we sort its elements into an array $\mathcal{T}_A$. Then, we perform a binary search on the array $\mathcal{T}_A$ to find the optimal period, testing at each step whether the current element T is a feasible value. To do so, we use the greedy assignment procedure of Algorithm 1. Initially, the current element T is the median of $\mathcal{T}_A$. If the greedy assignment procedure returns "failure", we increase the period by jumping to the median of the elements of $\mathcal{T}_A$ which are larger than T, and if it returns "success", we jump to the median of the elements of $\mathcal{T}_A$ which are smaller than T. The algorithm terminates in $\lceil \log \mathcal{T} \rceil$ iterations.

Algorithm 1 Greedy-Assignment(T)

```
Work with fastest $N$ processors, numbered $P_1$ to $P_N$, where $s_1 \le s_2 \le \cdots \le s_N$
Mark all stages $\mathcal{S}_1$ to $\mathcal{S}_N$ as free
for $u = 1$ to $N$ do
  Pick up any free stage $\mathcal{S}_a^k$ such that:
    $W_a \times \max(\frac{\delta_a^{k-1}}{b_a}, \frac{w_a^k}{s_u}, \frac{\delta_a^k}{b_a}) \le T$
  Assign $\mathcal{S}_a^k$ to $P_u$
  Mark $\mathcal{S}_a^k$ as already assigned
  if no stage found then
    return "failure"
  end if
end for
return "success"
```

Note that $|\mathcal{T}| \le N \times p$ (N stages and p processors), hence the total computation time is $O((N \times p + cost_{GA}) \log(N \times p))$, where $cost_{GA}$ is the cost of the greedy assignment procedure.

We now describe the greedy assignment algorithm for a prescribed value T of the achievable period. Recall that there are N stages to map onto $p \ge N$ processors in a one-to-one fashion. Also, we target communication homogeneous platforms with different-speed processors ($s_u \ne s_v$), with different-capacity links between the applications, but with links of same capacities within an application. First we retain only the fastest N processors, which we rename $P_1, P_2, \ldots, P_N$ such that $s_1 \le s_2 \le \cdots \le s_N$. Then we consider the processors in the order P_1 to P_N, i.e., from the slowest to the fastest, and greedily assign them any free (not already assigned) task that they can process within the period.

The proof that the greedy procedure returns a solution if and only if there exists a solution of period T is done by a simple exchange argument. Indeed, consider a valid one-to-one assignment of period T, denoted $\mathcal{A}$, and assume that it has assigned stage $\mathcal{S}_{a_1}^{k_1}$ to P_1. Note first that the greedy procedure will indeed find a stage to assign to P_1 and cannot fail, since $\mathcal{S}_{a_1}^{k_1}$ can be chosen. If the choice of the greedy procedure is actually $\mathcal{S}_{a_1}^{k_1}$, we proceed by induction with P_2. If the greedy procedure has selected another stage $\mathcal{S}_{a_2}^{k_2}$ for P_1, we find which processor, say P_u, has been assigned this stage in the valid assignment $\mathcal{A}$. Then we exchange the assignments of P_1 and P_u in $\mathcal{A}$. As P_u is faster than P_1, which could process $\mathcal{S}_{a_1}^{k_1}$ in time in the assignment $\mathcal{A}$, P_u can process $\mathcal{S}_{a_1}^{k_1}$ in time too.

As $\mathcal{S}_{a_2}^{k_2}$ has been mapped on P_1 by the greedy procedure,

P_1 can process $\mathcal{S}_{a_2}^{k_2}$ in time. So the exchange is valid, we can consider the new assignment which is valid and which did the same assignment on P_1 than the greedy procedure. The proof proceeds by induction with P_2 as before.

The complexity of the greedy assignment procedure is $cost_{GA} = O(N^2)$, because of the two loops over processors and stages. Altogether, since $N \leq p$, the cost of Algorithm 1 can be neglected, and the complexity of the whole algorithm is $O((N \times p)\log(N \times p))$, which is indeed polynomial in the problem size.

In addition we can observe that this algorithm works with the no-overlap communication model, by replacing $W_a \times \max(\frac{\delta_a^{k-1}}{b_a}, \frac{w_a^k}{s_u}, \frac{\delta_a^k}{b_a}) \leq T$ by $W_a \times (\frac{\delta_a^{k-1}}{b_a} + \frac{w_a^k}{s_u} + \frac{\delta_a^k}{b_a}) \leq T$. ■

Theorem 2: On fully heterogeneous platforms, the problem of finding a one-to-one mapping that minimizes the period is NP-complete.

Proof: As the problem was already NP-complete with one single application [12], it remains NP-complete with concurrent applications. ■

B. Interval mapping

Theorem 3: On fully homogeneous platforms, an interval mapping that minimizes the period can be determined in polynomial time.

Proof: A polynomial algorithm has already been found to exhibit the minimal period with one application, under a communication model without overlap [12], and it can easily be extended to the overlap model, so the following proof is valid for both models. We exhibit an algorithm (see Algorithm 2) which finds an optimal interval mapping for concurrent applications, thanks to the previous polynomial algorithm for a single application, and we show its validity.

Algorithm 2

```
Assign all stages of each application to one processor
Compute the period of all applications
for $a = (p - A)$ to $p$ do
  Find an application $a'$ such that $W_{a'} \times T_{a'}$ is maximum
  Add one processor to this application
  Compute the new period $T_{a'}$ of this application
end for
```

First, here are some notations:

- $(k_{a,i}^u)$ is a A-tuple which represents the processor distribution among the applications at step i of Algorithm 2.
- $(k_{a,i}^o)$ is an optimal processor distribution with i processors.
- $T_a(n)$ is the period of the application numbered a, where n is the number of processors the application a is assigned to.
- $T(d) = \max_{a\in\{1,\dots,A\}} W_a \times T_a(d_a)$, where d is a A-tuple.

Let us prove now the optimality of Algorithm 2.

- $(k_{a,A}^u)$ is the best distribution with A processors, because it is the only one.
- Let us assume that $(k_{a,i}^u)$ is optimal with i processors. We want $(k_{a,i+1}^u)$ to be an optimal distribution with $i+1$ processors.
  - Either: $\exists a, k_{a,i+1}^o < k_{a,i}^u$
    In this case, by construction,
    $$\exists i' < i, T((k_{a,i'}^u)) = W_a \times T_a(k_{a,i'}^u) = W_a \times T_a(k_{a,i+1}^o)$$
    Now, because every T_a and $x \mapsto W_a \times x$ are non-decreasing, $T((k_{a,i+1}^u)) \leq T((k_{a,i}^u)) \leq T((k_{a,i'}^u))$, and by definition $W_a \times T_a(k_{a,i+1}^o) \leq T((k_{a,i+1}^o))$.
    Finally, $T((k_{a,i+1}^u)) \leq T((k_{a,i+1}^o))$.
  - Or: $\exists! a, k_{a,i+1}^o = k_{a,i}^u + 1$
    - either: $k_{a,i+1}^u = k_{a,i}^u + 1$ and we are done,
    - or: $\exists a' \neq a, k_{a',i+1}^u = k_{a',i}^u + 1$
      In this case, by construction,
      $T((k_{a,i}^u)) = f_{a'}(T_{a'}(k_{a',i}^u)) = f_{a'}(T_{a'}(k_{a',i+1}^o))$ because $k_{a',i}^u = k_{a',i+1}^o$. Thus $T((k_{a,i}^u)) \leq T((k_{a,i+1}^o))$. Finally, $T((k_{a,i+1}^u)) \leq T((k_{a,i+1}^o))$.

  Overall we have shown that $(k_{a,i+1}^u)$ was as good as $(k_{a,i+1}^o)$.
- By induction, the algorithm finds an optimal solution to map A applications onto p processors.

The complexity of computing the period of application a with $q \leq p$ processors, keeping the intermediate result with $q - 1$ processors, is bounded by $O((n_a)^3 q)$ [12]. Let $n_{max} = \max_{a\in\{1,\dots,A\}} n_a$. Since we perform at most p steps in the algorithm, and $q \leq p$, the complexity of Algorithm 2 is bounded by $O(n_{max}^3 p^2)$, which is indeed polynomial in the problem size. ■

Theorem 4: On communication homogeneous platforms, the problem of finding an interval mapping that minimizes the period is NP-complete.

Proof: As the problem was already NP-complete with one single application [12], it remains NP-complete with concurrent applications. ■

The case **special-app** is more interesting, because a polynomial algorithm exists to find an interval mapping which minimizes the period of one single application [13]; however, the problem becomes NP-complete with several applications.

Theorem 5: With more than one application, heterogeneous processors, homogeneous pipelines without communication, the problem of finding an interval mapping which minimizes $\max_{a\in\{1,\dots,A\}} T_a$ is NP-complete (in the strong sense).

Proof: We consider the associated decision problem: given a period T, is there a mapping of period less than T? The problem is obviously in NP: given a period and a mapping, it is easy to check in polynomial time that it is valid by computing its period.

To establish the completeness, we use a reduction from 3-PARTITION [23]. We consider an instance $\mathcal{I}_1$ of 3-PARTITION: given an integer B and $3m$ positive integers $a_1, a_2, \dots, a_{3m}$ such that for all $i \in \{1, \dots, 3m\}$, $B/4 < a_i < B/2$ and with $\sum_{i=1}^m a_i = mB$, does there exist a partition $I_1, \dots, I_m$

of $\{1, \ldots, 3m\}$ such that for all $j \in \{1, \ldots, m\}$, $|I_j| = 3$ and $\sum_{i \in I_j} a_i = B$?

As 3-PARTITION is NP-complete in the strong sense, we can encode the $3m$ numbers in unary, and assume that the size of $\mathcal{I}_1$ is $O(mB)$.

We build an instance $\mathcal{I}_2$ of our problem with m identical applications such that each application is composed of B stages, with $w = 1$, and $p = 3m$ processors with speeds a_j for each $j \in \{1, \ldots, 3m\}$. We ask whether it is possible to realize a period of 1. Clearly, the size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$ (coded in unary). We now show that instance $\mathcal{I}_1$ has a solution if and only if instance $\mathcal{I}_2$ does.

Suppose first that $\mathcal{I}_1$ has a solution. Let $I_j = \{a'_{1,j}, a'_{2,j}, a'_{3,j}\}$, for $j \in \{1, \ldots, m\}$. For each $j \in \{1, \ldots, m\}$, we assign the $a'_{1,j}$ first consecutive stages of the application j to the processor of speed $a'_{1,j}$, the $a'_{2,j}$ next stages to the processor of speed $a'_{2,j}$, and the $a'_{3,j}$ remaining stages to the processor of speed $a'_{3,j}$. As the period of every processor is clearly equal to 1, the period is 1.

Suppose now that $\mathcal{I}_2$ has a solution. As the sum of all computation times is equal to the sum of all processor speeds, and a processor cannot be assigned stages of two different applications, for each application, the sum of its computation times is equal to the sum of the speed of processors which are assigned a stage of this application. Now, for all $i \in \{1, \ldots, 3m\}$, $B/4 < a_i < B/2$, so there are exactly three processors involved in the processing of each application. We can derive easily a solution to $\mathcal{I}_1$ (set I_j corresponding to processors of application j).

As there is no communication, this proof is valid for both communication models. ∎

Theorem 6: With more than one application, heterogeneous processors, homogeneous pipelines without communication, the problem of finding the optimal interval mapping which minimizes $\max_{a \in \{1,\ldots,A\}} W_a \times T_a$ is NP-complete (in the strong sense).

Proof: We follow the previous proof, but we assume now that, for each $a \in \{1, \ldots, A\}$, for $k \in \{1, \ldots, m\}$, $w_a^k = 1/W_a$. Then we scale each application: each w_a^k is multiplied by W_a so that the new period T'_a of the application a will be $W_a T_a$. We are now in the case of the previous theorem. ∎

Theorem 7: With more than one application, heterogeneous processors, homogeneous pipelines without communication, the problem of finding the optimal interval mapping which minimizes $\max_{a \in \{1,\ldots,A\}} T_a / T_a^*$ is NP-complete (in the strong sense).

Proof: We build the same instance as the one of the first proof. As the pipeline applications are all similar, the period of those applications when they are alone on the platform are all the same. We finally just have to minimize $\max_{a \in \{1,\ldots,A\}} T_a$. ∎

V. Complexity of multi-criteria problems

When dealing with multiple criteria, our approach is to minimize one of them, given a threshold on the others. Actually, fixing the period or the latency means fixing a threshold on the period or latency of each application, thus providing a table of period or latency values. Equivalently, we minimize the value of Equation (6) with suitable coefficients. For the energy, only a bound on the global energy consumption is required. All complexity results are summarized in Table II. Just as before, all results apply to both the overlap and no-overlap models, and to all objective functions introduced in Section III-C. As shown in Table II, we consider the three following combinations: period/latency, period/energy and period/latency/energy. In this section, we briefly summarize results for the period/latency combination, and we put an emphasis on the last two combinations involving both performance and energy criteria.

A. Period/latency minimization

In this section again, we are not concerned with energy minimization issues, so, similarly to results of Section IV, all processors can be run systematically at their highest speed. Therefore, on fully homogeneous platforms, all one-to-one mappings are identical, and it is straightforward to minimize the latency for a given period, or the converse.

However, for interval mappings, we must decide where to split applications into intervals, and we provide a dynamic programming algorithm which solves both variants of the problem with a single application. When considering multiple applications, we need to run the dynamic programming algorithm once per application with the corresponding period (resp. latency) threshold, and the minimum latency (resp. period) that can then be achieved is the maximum over all applications. Please see [22] for details. When moving to a platform with heterogeneous processors, even if the application is homogeneous with no communication (case **special-app**), the problem of finding a one-to-one or interval mapping that solves the bi-criteria period/latency problem is NP-complete. This result is a direct consequence of the NP-completeness of the mono-criterion cases, see Section IV.

B. Period/energy minimization

We first provide results for one-to-one mappings, and then discuss interval mappings. For fully heterogeneous platforms, the problem is NP-hard because the period minimization problem already is NP-hard on such platforms. The interesting result is the following:

Theorem 8: On communication homogeneous platforms, a one-to-one mapping which minimizes the energy consumption while enforcing a given period for each application can be determined in polynomial time.

Proof: We build a bipartite graph $G = (U, V, E)$, and prove that the problem amounts to finding a minimum weighted matching in this graph. U is the processor set, and V the stage set. For each processor and each stage, the weight of the edge between the two vertices is set to $+\infty$ if the processor cannot execute the stage within the period, and else it is the energy consumed by the processor when it is running in the smallest mode allowing to execute the stage within the period.

	proc-hom	**proc-het**		
	com-hom	**special-app**	**com-hom**	**com-het**
Period/Latency - *both*	polynomial	NP-complete		
Period/Energy - *one-to-one*	polynomial (minimum matching)			NP-complete
Period/Energy - *interval*	polynomial (dyn. prog.)	NP-complete		
Period/Latency/Energy - *both*	NP-complete			

Table II
COMPLEXITY RESULTS FOR MULTI-CRITERIA OPTIMIZATION PROBLEMS.

Finding a minimum weighted matching gives us the minimum power consumption, in polynomial time $O\left((N+p)^{\frac{3}{2}}\right)$. ■

For interval mappings, first note that the problem becomes NP-complete as soon as we consider different speed processors, because of the NP-completeness of the period minimization problem for such platforms. Thus we focus on fully homogeneous platforms.

Theorem 9: On fully homogeneous platforms, an interval mapping which minimizes the energy consumption while enforcing a given period for each application can be determined in polynomial time.

Proof: We first exhibit a dynamic programming algorithm that returns the optimal energy consumption for a single application, when using exactly k processors to compute the application. This algorithm is fixing the processor speeds so as to minimize the energy. Then, the multiple application case can be solved using another dynamic programming algorithm, which decides how many processors should be allocated to each application.

For a single application $a \in \{1,\dots,A\}$, and a processor number $q \in \{1,\dots,p\}$, we compute E_a^q, the minimum energy consumed for the application a using at most q processors. We recursively compute the value $E(i,j,k)$, which is the optimal energy consumption that can be achieved by any interval-based mapping of stages $\mathcal{S}_a^i$ to $\mathcal{S}_a^j$ using exactly k processors. The goal is to determine $E_a^q = \min_{k\in\{1,\dots,q\}} E(1,n_a,k)$. The recurrence relation can be expressed as:

$$E(i,j,k) = \min_{i\le\ell\le j-1} \left(E(i,\ell,k-1) + E(\ell+1,j,1)\right)$$

with the initialization:

- $E(i,i,r) = +\infty \quad \text{if} \quad r > 1$
- Defining

$$\mathcal{F}_i^j = \Bigg\{ E_{dyn}(s_\ell) + E_{stat},$$
$$\max\left(\frac{\delta^{i-1}}{b}, \frac{\sum_{k=i}^{j} w^k}{s_\ell}, \frac{\delta^j}{b}\right) \le T, \ell \in \{1,\dots,m\}\Bigg\},$$

we have:

$$E(i,j,1) = \begin{cases} \min \mathcal{F}_i^j & \text{if} \;\; \mathcal{F}_i^j \neq \varnothing \\ +\infty & \text{otherwise} \end{cases}$$

Here, m is the number of speed modes, and T is the period bound for the application a. The complexity of this dynamic programming algorithm is bounded by $O(n_a^2(p+m))$.

Note that for the no-overlap model, we simply replace $\max\left(\frac{\delta^{i-1}}{b}, \frac{\sum_{k=i}^{j} w^k}{s_\ell}, \frac{\delta^j}{b}\right)$ by $\frac{\delta^{i-1}}{b} + \frac{\sum_{k=i}^{j} w^k}{s_\ell} + \frac{\delta^j}{b}$ in the definition of $\mathcal{F}_i^j$.

Note also that $E_a^k = +\infty$ if the algorithm fails to match the period T.

For several applications, let $E(a,k)$ the minimum energy consumed by k processors on the first applications $1,\dots,a$, so we are looking for $E(A,p)$. This energy can be computed recursively, thanks to the recurrence relation:

$$\forall k \in \{1,\dots,p\}, \forall a \in \{2,\dots,A\},$$
$$E(a,k) = \min_{q\in\{0,\dots,k-1\}} \left(E_a^q + E(a-1,k-q)\right)$$

and the initialization:

$$\forall k \in \{1,\dots,p\}, \quad E(1,k) = E_1^k$$

The overall complexity is $O(AN^3p^2)$. ■

C. Period/latency/energy minimization

When mixing the three criteria, the problem becomes NP-hard even for fully homogeneous platforms, no communication, and a single application. The combinatorial nature of the problem comes from the fact that even if processors are identical, they are multi-modal and each of them may run at a different speed.

Theorem 10: On fully homogeneous platforms, with a single application and without any communication cost, finding a one-to-one mapping that solves the tri-criteria problem is NP-hard.

Proof: We consider the associated decision problem: given a period T, a latency L and an energy E, does there exist a one-to-one mapping of period less than T, latency less than L and energy less than E?

The problem is obviously in NP: given a period, a latency, an energy and a mapping, it is easy to check in polynomial time that the mapping is valid.

To establish the completeness, we use a reduction from 2-PARTITION [23]. We consider an instance $\mathcal{I}_1$ of 2-PARTITION: given n strictly positive integers $a_1, a_2, \dots, a_n$, does there exists a subset I of $\{1,\dots,n\}$ such that $\sum_{i\in I} a_i = \sum_{i\notin I} a_i$? Let $S = \sum_{i=1}^{n} a_i$. Let $K = \alpha \times S + 2$, where α is the exponent used in the computation of the energy (see Section III-E).

We build an instance $\mathcal{I}_2$ of our problem with n identical processors, each with $m = 2n + 1$ modes such that:

$$\forall i \in \{1, \ldots, n\} \quad \begin{cases} s_{2i-1} = K^i \\ s_{2i} = K^i + \frac{a_i X}{K^{i(\alpha-1)}} \end{cases}$$

and a pipelined application composed of n stages, with computation costs $w_i = K^{i(\alpha+1)}$.

Intuitively, the idea is to choose K such that:

1) stage weights are far enough from one another;
2) there is a gap between (s_{2i-1}, s_{2i}) and (s_{2j-1}, s_{2j}).

Then the mapping will use exactly one component of every pair (s_{2i-1}, s_{2i}).

We claim that for each $j \in \{2, \ldots, n\}$, we have

$$K^{j\alpha} > \sum_{i=1}^{j-1} K^{i\alpha} + \alpha \left(\frac{S}{2} - \frac{1}{2} \right)$$

and

$$K^{j\alpha+1} > \sum_{i=1}^{j} K^{i\alpha} + \left(K^{1-\alpha} \times a_{j-1} + 1 - \frac{S}{2} \right).$$

To prove the claim, let $j \in \{2, \ldots, n\}$. On the one side:

$$\begin{aligned} &\sum_{i=1}^{j-1} K^{i\alpha} + \alpha \left(\frac{S}{2} - \frac{1}{2} \right) \\ &\qquad < \sum_{i=1}^{j-1} K^{i\alpha} + \alpha S < (j-1)K^{(j-1)\alpha} + K \\ &\qquad < jK^{(j-1)\alpha} < K^{j\alpha} \end{aligned}$$

On the other side:

$$\begin{aligned} &\sum_{i=1}^{j} K^{i\alpha} + \left(K^{1-\alpha} \times a_{j-1} + 1 - \frac{S}{2} \right) \\ &\qquad < \sum_{i=1}^{j} K^{i\alpha} + K^{1-\alpha} \times K \\ &\qquad < jK^{j\alpha} + K^{2-\alpha} < (j+1)K^{j\alpha} < K^{j\alpha+1} \end{aligned}$$

For each $j \in \{2, \ldots, n\}$ and each $0 < X < 1$:

$$K^{j\alpha} > \sum_{i=1}^{j-1} K^{i\alpha} + \alpha X \left(\frac{S}{2} - \frac{1}{2} \right)$$

and

$$K^{j\alpha+1} > \sum_{i=1}^{j} K^{i\alpha} + X \left(K^{1-\alpha} \times a_{j-1} + 1 - \frac{S}{2} \right).$$

For all $i \in \{1, \ldots, n\}$, if we choose speed s_{2i} instead of speed s_{2i-1}, the additional energy is:

$$\begin{aligned} s_{2i}^\alpha - s_{2i-1}^\alpha &= (K^i + \frac{a_i X}{K^{i(\alpha-1)}})^\alpha - K^{i\alpha} \\ &= K^{i\alpha}(1 + \alpha \frac{a_i X}{K^{i\alpha}} + o(X)) - K^{i\alpha} \\ &= \alpha a_i X + f_i^E(X) \end{aligned}$$

where $f_i^E(X) \underset{x \to 0}{=} o(X)$.

In the same way, for each $i \in \{1, \ldots, n\}$, the difference in latency when using speed s_{2i} instead of speed s_{2i-1} to execute stage $\mathcal{S}_i$ is:

$$\begin{aligned} \frac{w_i}{s_{2i-1}} - \frac{w_i}{s_{2i}} &= \frac{K^{i(\alpha+1)}}{K^i} - \frac{K^{i(\alpha+1)}}{K^i + \frac{a_i X}{K^{i(\alpha-1)}}} \\ &= \frac{K^{i(\alpha+1)}}{K^i} - \frac{K^{i(\alpha+1)}}{K^i} \left(1 - \frac{a_i X}{K^{i\alpha}} + o(X) \right) \\ &= a_i X - f_i^L(X) \end{aligned}$$

where $f_i^L(X) \underset{x \to 0}{=} o(X)$.

For all $i \in \{2, \ldots, n\}$, the time to execute $\mathcal{S}_i$ at speed s_{2i-2} is:

$$\begin{aligned} \frac{w_i}{s_{2i-2}} &= \frac{K^{i(\alpha+1)}}{K^{i-1} + \frac{a_{i-1} X}{K^{(i-1)(\alpha-1)}}} \\ &= \frac{K^{i(\alpha+1)}}{K^{i-1}} \left(1 - \frac{a_{i-1} X}{K^{(i-1)\alpha}} + o(X) \right) \\ &= K^{i\alpha+1} - K^{1-\alpha} \times a_{i-1} X + f^{L_i}(X) \end{aligned}$$

So we choose $X < 1$ small enough, so that for each $i \in \{1, \ldots, n\}$,

$$\begin{cases} |f_i^E(X)| < X \times \frac{\alpha}{2n} \\ |f_i^L(X)| < X \times \frac{1}{2n} \end{cases}$$

and for all $i \in \{2, \ldots, n\}$, $|f^{L_i}(X)| < X \times \frac{1}{2}$.

Finally, we have to decide for the latency, the energy and the period bounds. Let E^* and L^* be the energy and latency obtained when $\mathcal{S}_i$ is executed at speed s_{2i-1} for all $i \in \{1, \ldots, n\}$, $E^* = \sum_{i=1}^n s_{2i-1}^\alpha = \sum_{i=1}^n K^{i\alpha}$ and $L^* = \sum_{i=1}^n \frac{w_i}{s_{2i-1}} = E^*$. We ask whether it is possible to achieve an energy $E^o = E^* + \alpha X(S/2 + 1/2)$, a latency $L^o = L^* - X(S/2 - 1/2)$ and a period $T^o = L^o$.

Clearly, the size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$. We show that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ does.

Assume first that $\mathcal{I}_1$ has a solution. For each $i \in I$, stage $\mathcal{S}_i$ is executed at speed s_{2i}, and for each $i \in \{1, \ldots, n\} \setminus I$, stage $\mathcal{S}_i$ is executed at speed s_{2i-1}.

The mapping consumes an energy E and has a latency L, where:

$$\begin{aligned} E &= E^* + \sum_{i \in I} \left(s_{2i}^\alpha - s_{2i-1}^\alpha \right) = E^* + \sum_{i \in I} \left(\alpha a_i X + f_i^E(X) \right) \\ &\leq E^* + \sum_{i \in I} \left(\alpha a_i X + \frac{\alpha X}{2n} \right) \leq E^* + \alpha X \left(\frac{S}{2} + \frac{1}{2} \right) \\ &\leq E^o \end{aligned}$$

$$\begin{aligned} L &= L^* - \sum_{i \in I} \left(\frac{w_i}{s_{2i-1}} - \frac{w_i}{s_{2i}} \right) = L^* - \sum_{i \in I} \left(a_i X - f_i^L(X) \right) \\ &\leq L^* - \sum_{i \in I} \left(a_i X - \frac{X}{2n} \right) \leq L^* - X \left(\frac{S}{2} - \frac{1}{2} \right) \\ &\leq L^o \end{aligned}$$

Because $T^o = L^o$, and because we fulfill the latency constraint, we fulfill the period constraint too. We conclude that $\mathcal{I}_2$ has a solution.

Suppose now that $\mathcal{I}_2$ has a solution. We first show that for each $i \in \{1, \dots, n\}$, stage $\mathcal{S}_i$ is executed at speed either s_{2i-1} or s_{2i}. Let $(\mathcal{P}_j)$ be the property: for each $i \in \{j, \dots, n\}$, there is a single processor running at speed s_{2i-1} or s_{2i}, and this processor is assigned stage $\mathcal{S}_i$. We first prove that $(\mathcal{P}_n)$ is true. On the one hand, if two processors were running at speed s_{2n-1} or s_{2n}, they would consume an energy

$$E \geq 2s_{2n-1}^{\alpha} > K^{n\alpha} + \sum_{i=1}^{n-1} K^{i\alpha} + \alpha X \left(\frac{S}{2} + \frac{1}{2}\right) > E^o$$

On the other hand, if no processor was running at speed s_{2n-1} or s_{2n}, the latency would verify

$$\begin{aligned} L \geq & \quad \frac{w_n}{s_{2n-2}} \geq K^{n\alpha+1} - K^{1-\alpha} \times a_{n-1}X + f^{L_i}(X) \\ > & \quad \sum_{i=1}^{n} K^{i\alpha} + X\left(K^{1-\alpha} \times a_{n-1} + 1 - \frac{S}{2}\right) \\ & \qquad - K^{1-\alpha} \times a_{n-1}X + f^{L_i}(X) \\ > & \quad \sum_{i=1}^{n} K^{i\alpha} - X\left(\frac{S}{2} - \frac{1}{2}\right) + \left(\frac{X}{2} + f^{L_i}(X)\right) \\ > & \quad L^o \end{aligned}$$

We conclude that $(\mathcal{P}_n)$ is true. We now proceed by induction. If for some $j \in \{3, \dots, n\}$, $(\mathcal{P}_j)$ is true, then we show that $(\mathcal{P}_{j-1})$ is true in a quite similar way. In the end, $(\mathcal{P}_2)$ is true (and the processor that is assigned stage $\mathcal{S}_1$ is running either at speed s_1, or at speed s_2). Let I the subset of $\{1, \dots, n\}$ such that the processor that is assigned the stage $\mathcal{S}_i$ is running at speed s_{2i}. Then for each $i \in \{1, \dots, n\} \backslash I$, the processor that is assigned stage $\mathcal{S}_i$ is running at speed s_{2i-1}. The consumed energy is

$$E = E^* + \sum_{i \in I} \left(\alpha a_i X + f_i^E(X)\right)$$

But $E \leq E^o$, hence

$$\sum_{i \in I} a_i \leq \frac{S}{2} + \left(\frac{1}{2} - \frac{\sum_{i \in I} f_i^E(X)}{\alpha X}\right)$$

Therefore

$$\sum_{i \in I} a_i < \frac{S}{2} + \left(\frac{1}{2} + \frac{1}{2}\right).$$

As the a_i are integers, we derive that $\sum_{i \in I} a_i \leq \frac{S}{2}$.

The achieved latency is $L = L^* - \sum_{i \in I}\left(a_i X - f_i^L(X)\right)$, and $L \leq L^o$, hence

$$\sum_{i \in I} a_i \geq \frac{S}{2} - \left(\frac{1}{2} - \frac{\sum_{i \in I} f_i^L(X)}{X}\right)$$

Since $\frac{\sum_{i \in I} f_i^L(X)}{X} \leq \frac{1}{2}$, we get $\sum_{i \in I} a_i \geq \frac{S}{2}$.

Finally, $\sum_{i \in I} a_i = \frac{S}{2}$ and $\mathcal{I}_1$ has a solution, which concludes the proof. ■

Theorem 11: On fully homogeneous platforms, with a single application and without any communication cost, finding an interval mapping that solves the tri-criteria problem is NP-hard.

Proof: We only give the sketch of the completeness proof, which reuses the proof of Theorem 10. To construct the instance $\mathcal{I}_2$, we insert big stages between the previous stages. We add a big speed to the processor modes, adjusted to allow the execution of exactly one big stage during the period. More formally, we build a pipeline composed of $2n-1$ stages, such that $\forall i \in \{1, \dots, n\}, w_{2i-1} = K^{i(\alpha+1)}$ and $\forall i \in \{1, \dots, n-1\}, w_{2i} = K^{(n+1)(\alpha+1)}$ We use $2n-1$ identical processors, that can run $2n+1$ modes, such that $\forall i \in \{1, \dots, n\}, s_{2i-1} = K^i$ and $s_{2i} = K^i + \frac{a_i X}{K^{i\alpha}}$. We also let $s_{2n+1} = K^{n+1}$.

We search for an interval mapping, whose energy does not exceed $E^o = (n-1)K^{(n+1)\alpha} + E^* + \alpha X(S/2 + 1/2)$, whose latency does not exceed $L^o = (n-1)K^{(n+1)\alpha} + L^* - X(S/2 - 1/2)$, and whose period does not exceed $T^o = K^{(n+1)\alpha}$. If the instance $\mathcal{I}_1$ of 2-PARTITION has a solution, we proceed like in the previous proof, and map every big stage onto a processor that is running in its highest mode. All constraints are fulfilled.

If the instance $\mathcal{I}_2$ has a solution, we have to run processors that are assigned a big stage in their highest mode. Moreover, these processors cannot be assigned other stages. All we have to do next is to find a one-to-one mapping of the unassigned stages, with the additional constraint that we cannot run the remaining processors in their highest modes without exceeding the energy bound. We then conclude as in the proof of Theorem 10. ■

We conclude this section with some remarks on uni-modal processors. If we restrict to processors with a single execution mode, the problem becomes polynomial on fully homogeneous platforms, while it remains NP-hard otherwise (because of the NP-completeness of the period/latency problem which is also established with uni-modal processors). For one-to-one mappings, all mappings are equivalent on fully homogeneous platforms, but the algorithm is more sophisticated for interval mappings. We first write an algorithm which partitions the stages of a single application into intervals, for each of the three variants of the tri-criteria optimization problem, and then we use this algorithm for the multiple application problem. Details can be found in [22].

VI. Conclusion

In this paper, we have studied the problem of mapping concurrent applications onto computational platforms according to three criteria: period, latency and energy. We restricted the study to the class of applications which have a pipeline structure, and we established the complexity of the problems for different variants of mapping strategies (one-to-one and interval mappings), and different types of platforms (ranking from fully homogeneous to fully heterogeneous).

First we considered performance criteria, namely period or latency minimization. From this study of mono-criterion problems, one striking result is the impact of having multiple concurrent applications on the problem complexity. Indeed, when several applications are in competition for resources, the period minimization problem turns out NP-hard for interval mappings with heterogeneous processors, homogeneous pipelines and without communication, while a polynomial algorithm had been found to solve the same problem with a single application. The same phenomenon happens for latency minimization with one-to-one mappings. For other period or latency minimization problems, either we were able to extend polynomial algorithms for the single application case, or the problem remained NP-complete. Considering bi-criteria problems, we put a particular emphasis on problems involving both performance and energy criteria. We were able to derive nice sophisticated multi-criteria polynomial algorithms, through the construction of bipartite graphs or the use of dynamic programming. Trade-offs were found to allow for an efficient albeit energy-aware execution. Finally, the most challenging tri-criteria problem period/latency/energy turned out to be NP-hard even with a single application on a fully homogeneous platform and no communication cost.

We believe that this exhaustive complexity analysis provides a solid theoretical foundation for the study of multi-criteria mappings of several concurrent applications, in particular when combining performance and energy optimization criteria.

As future work, on the theoretical side, we envision to add replication into the mappings: a stage could be mapped onto several processors, each in charge of different data sets, in order to improve the period, as was investigated in [13]. The problem would become even more challenging in a framework accounting for energy issues. On a more practical side, we plan to design some polynomial-time heuristics to solve the tri-criteria optimization problem in a general framework, in order to offer practical solutions to a difficult problem. It would also be challenging, both from a theoretical and a practical perspective, to assess the impact of processor sharing between applications, for situations in which it would be allowed by the platform manager.

Acknowledgment

A. Benoit and Y. Robert are with the Institut Universitaire de France. This work was supported in part by the ANR *StochaGrid* project.

References

[1] DataCutter, "DataCutter Project: Middleware for Filtering Large Archival Scientific Datasets in a Grid Environment," http://www.cs.umd.edu/projects/hpsl/ResearchAreas/DataCutter.htm.

[2] K. Taura and A. A. Chien, "A heuristic algorithm for mapping communicating tasks on heterogeneous resources," in *Heterogeneous Computing Workshop*. IEEE Computer Society Press, 2000, pp. 102–115.

[3] Q. Wu and Y. Gu, "Supporting distributed application workflows in heterogeneous computing environments," in *International Conference on Parallel and Distributed Systems (ICPADS)*, Washington, DC, USA, 2008, pp. 3–10.

[4] S. L. Hary and F. Özgüner, "Precedence-constrained task allocation onto point-to-point networks for pipelined execution," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 10, no. 8, pp. 838–851, 1999.

[5] Q. Wu, J. Gao, M. Zhu, N. Rao, J. Huang, and S. Iyengar, "On optimal resource utilization for distributed remote visualization," *IEEE Transactions on Computers (TC)*, vol. 57, no. 1, pp. 55–68, 2008.

[6] M. Cole, "Bringing Skeletons out of the Closet: A Pragmatic Manifesto for Skeletal Parallel Programming," *Parallel Computing*, vol. 30, no. 3, pp. 389–406, 2004.

[7] F. Rabhi and S. Gorlatch, *Patterns and Skeletons for Parallel and Distributed Computing*. Springer Verlag, 2002.

[8] J. Subhlok and G. Vondran, "Optimal mapping of sequences of data parallel tasks," in *Principles and Practice of Parallel Programming (PPoPP)*, 1995.

[9] ——, "Optimal latency-throughput tradeoffs for data parallel pipelines," in *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 1996.

[10] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Saddayappan, and J. Saltz, "Toward optimizing latency under throughput constraints for application workflows on clusters," in *Euro-Par'07*, ser. LNCS 4641. Springer Verlag, 2007, pp. 173–183.

[11] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Sadayappan, and J. Saltz, "A duplication based algorithm for optimizing latency under throughput constraints for streaming workflows," in *International Conference on Parallel Processing*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 254–261.

[12] A. Benoit and Y. Robert, "Mapping pipeline skeletons onto heterogeneous platforms," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 68, no. 6, pp. 790–808, 2008.

[13] ——, "Complexity results for throughput and latency optimization of replicated and data-parallel workflows," *Algorithmica*, 2009, available online at http://dx.doi.org/10.1007/s00453-008-9229-4.

[14] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters," in *Proceedings of the ACM/IEEE conference on SuperComputing (SC)*. IEEE Computer Society, 2005, p. 34.

[15] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: modeling and implementation," *ACM Transactions on Architecture and Code Optimization*, vol. 1, no. 1, pp. 94–125, 2004.

[16] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi, "Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster," vol. 0, Los Alamitos, CA, USA, 2006, p. 340.

[17] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *International Symposium on Low Power Electronics and Design (ISLPED)*. ACM Press, 1998, pp. 197–202.

[18] N. T. Karonis, B. Toonen, and I. Foster, "MPICH-G2: A grid-enabled implementation of the message passing interface," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 63, no. 5, pp. 551–563, 2003.

[19] K. Agrawal, A. Benoit, L. Magnan, and Y. Robert, "Scheduling algorithms for workflow optimization," in *Proceedings of IPDPS'2010*. IEEE CS Press, May 2010, also available as research report RR-LIP-2009-22 at http://graal.ens-lyon.fr/∼abenoit/.

[20] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan, "Flow and stretch metrics for scheduling continuous job streams," in *Proceedings of SODA'98*, 1998.

[21] A. Benoit, Y. Robert, and E. Thierry, "On the complexity of mapping linear chain applications onto heterogeneous platforms," *Parallel Processing Letters (PPL)*, vol. 19, no. 3, pp. 383–397, 2009.

[22] A. Benoit, P. Renaud-Goud, and Y. Robert, "Performance and energy optimization of concurrent pipelined applications," LIP, ENS Lyon, France, Research Report RR-LIP-2009-27, September 2009, available at http://graal.ens-lyon.fr/∼abenoit.

[23] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.